Memory access behavior analysis of NUMA-based shared memory programs

Jie Tao[*], Wolfgang Karl and Martin Schulz
LRR-TUM, Institut für Informatik, Technische Universität München, 80290 München, Germany
Tel: +49-89-289-{28397,28278,28399}; E-mail: {tao,karlw,schulzm}@in.tum.de

Abstract: Shared memory applications running transparently on top of NUMA architectures often face severe performance problems due to bad data locality and excessive remote memory accesses. Optimizations with respect to data locality are therefore necessary, but require a fundamental understanding of an application's memory access behavior. The information necessary for this cannot be obtained using simple code instrumentation due to the implicit nature of the communication handled by the NUMA hardware, the large amount of traffic produced at runtime, and the fine access granularity in shared memory codes. In this paper an approach to overcome these problems and thereby to enable an easy and efficient optimization process is presented. Based on a low-level hardware monitoring facility in coordination with a comprehensive visualization tool, it enables the generation of memory access histograms capable of showing all memory accesses across the complete address space of an application's working set. This information can be used to identify access hot spots, to understand the dynamic behavior of shared memory applications, and to optimize applications using an application specific data layout resulting in significant performance improvements.

1. Motivation

The development of parallel programs which run efficiently on parallel machines is a difficult task and takes much more effort than the development of sequential codes. A programmer has to consider communication and synchronization requirements, the complexity of data accesses, and the problem of partitioning work and data. Even after a program has been validated and produces correct results, a considerable amount of work has to be done in order to tune the parallel program to efficiently exploit the resources of the parallel machine.

This task becomes even more complicated on parallel machines with NUMA characteristics (Non Uniform Memory Access). Shared memory programs running on top of such machines often face severe performance problems due to bad data locality and excessive remote memory accesses. In this case, optimizations with respect to data locality are necessary for a better performance. The information required for data locality optimizations cannot be acquired easily as communication events are potentially very frequent, relatively short, fine-grained, and implicit.

In this paper, a comprehensive approach for an easy and efficient data locality optimization process is presented. This approach is based on a hardware monitoring concept which allows the acquisition of very fine-grained communication events. The information is being delivered to a visualization tool which enables the generation of memory access histograms capable of showing all memory accesses across the complete virtual address space of an application's working set. Using this graphical representation, the programmer can identify access hot spots, understand the dynamic behavior of shared memory applications, and optimize the program with an application specific data layout resulting in significant performance improvements.

The approach has been developed and evaluated on PC clusters with an SCI interconnection technology (Scalable Coherent Interface [3,5]). SCI supports memory-oriented transactions over a ringlet-based network topology, effectively supporting a distributed shared memory in hardware. In order to support shared memory programming on top of such a NUMA ar-

[*]Jie Tao is a staff member of Jilin University, China and is currently pursuing her Ph.D at the Technische Universität München, Germany.

chitecture, a software framework has been developed within the SMiLE project (Shared Memory in a LAN like Environment) which closes the semantic gap between the global view of the distributed physical memories in NUMA architectures and the global virtual memory abstraction required by shared memory programming models [8,17]. This framework supports, in principle, almost arbitrary shared memory programming models on top of the PC cluster [13] and thereby creates a flexible target platform for the presented monitoring approach.

The paper is organized as follows. The next section presents the SMiLE approach supporting shared memory programming on SCI-based PC-clusters. The SMiLE monitoring approach is being covered in Section 3. Section 4 describes the tool environment supporting the data locality optimization process. The paper concludes with a brief overview of related work in Section 5 and some final remarks in Section 6.

2. Shared memory in NUMA clusters

Cluster systems in combination with NUMA (Non-Uniform Memory Access) networks work on the principle of a global physical address space and enable each node to transparently access the memories on all other nodes within the system. They thereby form a favorable architectural tradeoff by combining the scalability and cost-effectiveness of standard clusters with a shared memory support close to CC-NUMA and SMP systems. However, in order to exploit this hardware support for shared memory environments, a software framework is required which closes the semantic gap between the distributed physical memories of NUMA machines and the global virtual memory abstraction required by shared memory programming models.

Such a shared memory framework based on SCI (Scalable Coherent Interface [3]), an IEEE-standardized [5] cluster interconnect with NUMA capabilities, has been developed within the SMiLE [7] (Shared Memory in a LAN-like Environment) project, which broadly investigates in SCI-based cluster computing. In addition, this framework, called HAMSTER [13] (Hybrid-dsm based Adaptive and Modular Shared memory archiTEctuRe), is capable of supporting in principle almost arbitrary shared memory programming models on top of a common hybrid-DSM core, the SCI Virtual Memory or SCI-VM [8,17]. This new type of DSM system establishes the global virtual address space required for applications by combining the present NUMA HW-DSM with lean software management. On top of this fundament, HAMSTER provides a large range of shared memory and cluster resource services, including an efficient synchronization module called SyncMod [18]. These services are designed in way that allows the implementation of programming models in a low-complex fashion, enabling the creation of as many different models as required or necessary for the intended target users and/or applications. In summary, HAMSTER enables the efficient exploitation of loosely coupled NUMA clusters for shared memory programming without binding users to a new, custom API.

Despite the efficient and direct utilization of the underlying HW-DSM and the lean implementation of the corresponding software components, it can be observed that applications often suffer from significant performance deficiencies. The reason for this can be found in excessive remote memory accesses which, despite SCI's extremely low latency, still can be up to an order of magnitude slower than local ones. It is therefore of great importance to study and optimize the locality behavior of shared memory applications in such NUMA scenarios since this will have a significant impact in the overall execution time and parallel efficiency of these codes.

3. Observing shared memory accesses

In order to optimize the runtime behavior of shared memory applications on top of such loosely coupled NUMA architectures, it is necessary to enable users to understand the memory access pattern of their applications and their impact with respect to the actual memory distributions. Depending on the application and its complexity, this can be a quite difficult and tedious task. Therefore it is of importance to support this process with appropriate tools. The basis for any of these, however, is the ability to monitor the memory access behavior on-line during the runtime of the application.

3.1. Challenges

The most severe problems connected with such a monitoring component stem from the fact that shared memory traffic by default is of implicit nature and performed at runtime through transparently issued loads and stores to remote data locations. Unlike in message passing systems, where explicite communication points are known and hence can be instrumented, in a shared memory environment other ways to track remote communication have to be found.

In addition, shared memory communication is very fine-grained (normally at word level). This also renders code instrumentation recording each global memory operation infeasible since it would slow down the execution significantly and thereby distort the final monitoring to a point where it is unusable for an accurate performance analysis. In addition, such an approach would require a complete cache and consistency model emulation since these two components play a large role in filtering the actual memory references which need to be served from main and/or remote memory.

Therefore, the only viable alternative is to deploy a hardware monitoring device observing the actual link traffic of the NUMA interconnection network. Only this guarantees fine-grained information about the actual communication after any access filtration by caches without influencing the actual execution behavior. Such a device, the SMiLE hardware monitor, has been developed within the SMiLE project for the observation of SCI network traffic [6].

3.2. *The SMiLE monitoring approach*

This SMiLE hardware monitor is designed to be attached to an internal link on current SCI adapters, the so-called B-Link. This link connects the SCI link chip to the PCI side of the adapter card and is hence traversed by all SCI transactions intended for or originating from the local node. Due to the bus-like implementation of the B-Link, these transactions can be snooped without influencing or even changing the target system and can then be transparently recorded by the SMiLE hardware monitor.

In order to prevent the necessity to actually store the complete transaction information for later processing and to enable an efficient on-line analysis of the observed data, the hardware monitor enables a sophisticated real-time analysis of the acquired data. The result are so-called *memory access histograms* which show the number of memory accesses across the complete virtual address space of an application's working set separated with respect to target node IDs. These histograms give the user a first and direct overview of the real memory access behavior of the target application.

In order to save valuable hardware resources, the SMiLE hardware monitor implements a swapping mechanism for its counter components. Whenever all counters are filled or one counter is about to overflow, a counter is evicted from the hardware monitor and stored in a ring buffer in main memory. The free counter is then reclaimed by the monitoring hardware for the further monitoring process. The resulting monitoring information in the main memory ring buffer is then collected by a corresponding driver software and combined to the complete access histogram. As the information, by the time it is evicted from the monitor, is relatively coarse grained, this combination step has rather low computational demands and therefore only a minimal impact on the application execution behavior [6].

In addition to this histogram mode, also referred to as dynamic mode due to its adaptability to the memory access behavior of the target application, the SMiLE monitor also contains a static monitoring component which allows to watch predefined events or accesses to special, user definable memory regions. In contrast to the former method, which is intended for a first performance overview of the complete application, this static mode is very useful for the detailed analysis of specific bottlenecks. Together, the two modes enable the complete analysis of the memory access behavior of shared memory applications on top of SCI.

Currently, the SMiLE hardware monitor is still under development with a first experimental prototype expected within the next six months. In order to still be able to start with software development and to prove the feasibility and usability of the presented approach, a realistic simulator for NUMA architectures has been developed [20] and is also being used within this work. It is designed in a way which allows a clean and easy migration from the simulation platform to the real hardware guaranteeing the validity of the presented approach.

3.3. *The SMiLE tool infrastructure*

In order to make the information gathered by the hardware monitor available to the user, it has to be first transformed to a higher level of abstraction. This is necessary since all acquired data is only based on individual memory accesses observed from the SCI network adapter and hence by nature based purely on physical addresses. This needs to be transformed into the virtual address space and then be related to source code information.

For this task, a comprehensive SMiLE software infrastructure is under development [9] (see also Fig. 1). It is based on the information acquired from the underlying monitoring component and transforms and enhances it using additional data collected from the various components of the overall system, including the SCI Virtual Memory and its synchronization module (see also Section 2) as well as various interfaces in programming models. This information is then aggregated and made globally accessible through the OMIS interface [1], an established on-line monitoring specification, and OCM (the OMIS Compliant Monitor) [23], the corresponding reference implementation.

As a result, this monitoring infrastructure enables a standardized and highly structured way for tools to access the distributed information. Currently the main focus lies on a sophisticated visualization tool, called DLV [21], which will be discussed below in more detail. In the future further shared memory tools will complete the monitoring infrastructure; especially an integrated and automatic data migration and load balancing component is envisioned.

4. Access behavior analysis

As mentioned in Section 2, the latency of remote memory accesses is one of the most important performance issues on NUMA systems. Optimizing programs with respect to data locality can minimize the accesses to remote memory modules and improve memory access performance. It requires, however, an understanding of a program's memory access behavior at run-time. The SMiLE tool infrastructure described in Section 3 provides a monitoring facility for observing the interconnection traffic and enables a comprehensive analysis of the runtime memory access behavior of shared memory applications based on the observed data.

4.1. The visualization tool

As already mentioned, the current focus of the SMiLE tool infrastructure lies on the *Data Layout Visualizer* (*DLV*) [21], a comprehensive visualization tool for shared memory traffic on NUMA architectures. It is capable of transforming the fine-grained data acquired by snooping hardware monitors like the SMiLE hardware monitor into a human-readable and easy-to-use form, enabling an exact understanding of an application's memory access behavior and the detection of communication hot spots. It provides a set of display windows showing the memory access histograms in various views and projecting the memory addresses back to data structures within the source code (see Fig. 2). This allows the programmer to analyze an application's access pattern and thereby forms the basis for any optimization of the physical data layout and distribution.

For this purpose, the DLV includes several different views on the acquired data, each presented by a specific window within the GUI. The most basic one among them is the "Run-time transfer" window (shown top left in Fig. 2). It is designed to illustrate a global overview of the actual data transfer performed on the interconnection fabric and visualizes the number of network packets between all nodes in the system. In addition, communication bottlenecks are highlighted based on user-defined thresholds (relative to a system wide average).

Going further into detail and looking at accesses related to their destination within the whole shared virtual space, the "Histogram table" (shown top right in Fig. 2) shows exact numbers of accesses at page granularity. As above, access hot spots are highlighted using user-defined thresholds and are also extracted into a further table shown in the "Most frequently accessed" window. The same information, however with less detail, can also be shown graphically in the "Access diagram" (shown at bottom of Fig. 2). It presents the memory access histogram at page granularity using colored columns to show the relative frequency of accesses performed by all nodes. In this diagram, inappropriate data allocation can be easily detected via the different heights of the columns. In addition, the corresponding data structure of a selected page can be shown in a small combined window using a mouse button within the graphic area. These diagrams therefore can direct users to place data on the correct node that mostly requires it.

The "Histogram table" and the "Access diagram" described above serve as the base for a correct allocation of pages accessed dominantly by one node. For pages accessed by multiple processors, however, it is more difficult to determine their location. For this purpose, the DLV provides a "Page analysis" window to illustrate the access behavior within a page, directing programmers to partition such a page and distribute it among the nodes in the system. An example for this is given in the next section.

Besides the direct information based on virtual addresses discussed so far, the DLV is also capable of relating the presented data to user data structures within the source code. This is done with the help of the "Data structure and location" window, which can be activated as a main window beyond other windows or as a subwindow within them, showing the mappings between

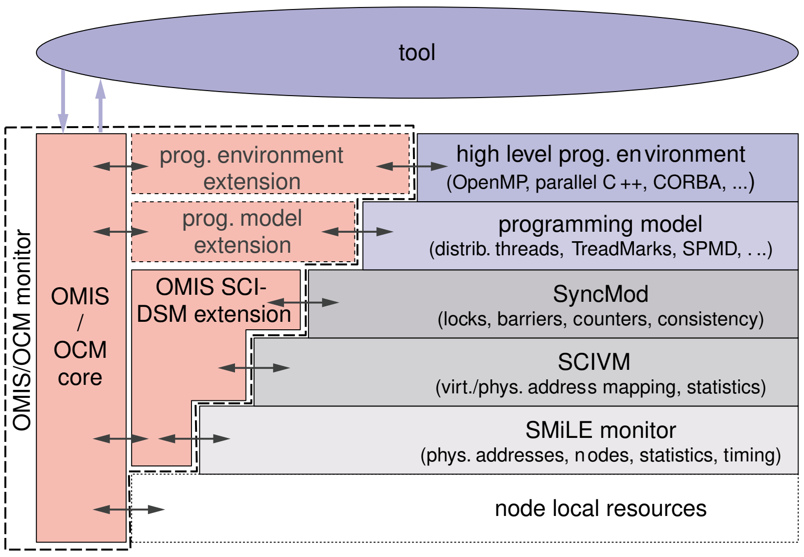

tool
OMIS/OCM monitor
OMIS / OCM core
prog. environment extension
prog. model extension
OMIS SCI-DSM extension
high level prog. environment (OpenMP, parallel C++, CORBA, ...)
programming model (distrib. threads, TreadMarks, SPMD, ...)
SyncMod (locks, barriers, counters, consistency)
SCIVM (virt./phys. address mapping, statistics)
SMiLE monitor (phys. addresses, nodes, statistics, timing)
node local resources

Fig. 1. Multilayer tool infrastructure.

Fig. 2. GUI of the *Data Layout Visualizer* (*DLV*) with a few sample windows.

the virtual address currently under investigation and the corresponding data structure within the source code. It therefore enables the user to relate the information acquired and visualized within the DLV to the data structures exhibiting the observed behavior and thereby to exactly modify the physical layout and distribution of the structures causing a problematic execution behavior.

4.2. *Analyzing a sample code*

The various DLV windows can efficiently be used to characterize the memory access behavior of shared memory codes. This will be demonstrated in the following using a Successive Over Relaxation (SOR) as a basic and easy-to-follow example. SOR is a numerical kernel that is used to iteratively solve partial differential

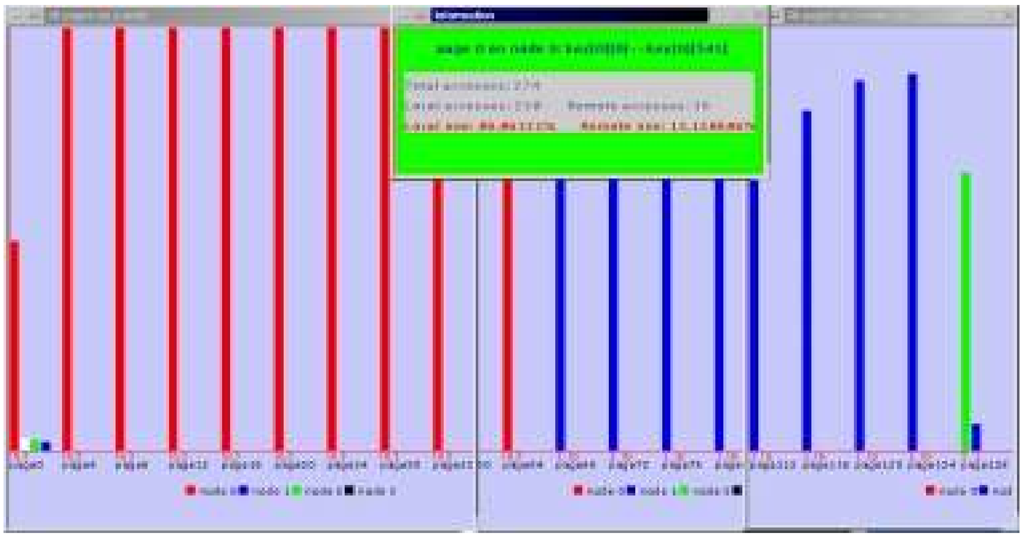


Fig. 3. Memory accesses of some pages on node 1 (SOR code).

equations. Its main working set is a large dense matrix array. During each iteration a four point stencil is applied to all points of this matrix. Due to the fixed and uniform work distribution across all matrix points and due to the numerical stability of the approach, which allows a reordering of the stencil updates, the matrix is split into blocks of equal size during the parallelization process. Each block is assigned to one processor of the cluster and each processor in the following only updates the part of the matrix assigned to it. On the other side, elements of the matrix array are stored transparently over the whole cluster corresponding to the default allocation scheme of the SCI-VM, a round-robin distribution at page granularity.

Processors therefore communicate on most matrix/memory accesses to access the data included in the individual blocks leading to poor speedup. In order to understand its memory access behavior, the execution of the SOR code, running on a 4-nodes cluster using a 512 by 512 grid (about 1 MB memory footprint), is simulated for 10 iterations and the monitored data is visualized. Based on this information, the access pattern of the SOR code can be exactly analyzed and in the next step optimized.

First we use the "Run-time transfer" window to get a first overview over the complete application and to detect simple communication bottlenecks. In this case, however, this display only shows that every node is accessed frequently by others without the existence of a single dominating bottleneck. This indicates that the overall working load of the SOR code is not allocated correctly. In order to find the hot spots, the next step is to analyze the memory access histogram using the "Access diagram" window. Figure 3 shows three different sections of the complete node diagram from the view of node 1 as the local node (incoming memory transactions). The memory access behavior of pages on other nodes is quite similar due to the symmetric work distribution and parallelization concept within SOR.

The figure first shows that page 0 behaves differently than the rest, as it is accessed by all nodes. By examining the "Data structure and location" information, it can be extracted that this is caused by accesses to global variables located at the beginning of this page before the actual matrix part. This includes information about the matrix size as well as IDs for nodes, locks, and barriers, which are required by all parallel threads. Past this initial information, it can be seen that all pages up to 64 are only accessed by node 1 (the local node), pages between 68 to 124 by node 2, and so on. The data structure information provided by the DLV (shown in the small top window in Fig. 3) additionally shows that these access blocks correspond to matrix blocks of the main SOR matrix.

While the "Access diagram" offers the general understanding of a program's access behavior, a more detailed analysis can be enabled using the "Page analysis" window of the visualizer. This window provides facilities to exhibit the memory accesses within a page and can be used for border pages with accesses from more than one node. Figure 4 shows the information about one such page: the "Section" subwindow, demonstrates the total references at finest possible granularity (mostly L2 cache line size) and the "Read/write" subwindow presents the concrete numbers of reads and writes. These windows thereby give exact information about the memory accesses within a page and can be used to clearly identify sharing properties.

In the concrete example, Fig. 4 shows that the first few sections are only accessed by node 1 followed by sections with overlapping accesses from both. This indicates that such a page can be partitioned and distributed with the first section located together with the whole block on pages 4 to 64 on the corresponding node. For the following section with true sharing, however, a correct distribution is more difficult. In this case, the Read/write curve can be used to determine the optimal placement: since blocking read operations are more expensive than writes, which are non-blocking, the node with a highest read frequency has the priority for owning them.

In summary, the information acquired from the DLV windows clearly shows the blocked memory access and matrix distribution strategy presented in the SOR code. This observation is also likely to hold for all other working set sizes. Hence, it is possible to deduce an application's memory access behavior based on the analysis of a single working set size and thereby to optimize the application in general.

4.3. Using the information for easy optimization

Based on the analysis described above, programs can be optimized by specifying a data layout fitting to their memory access pattern. This is done by placing data, which is indicated by the DLV as incorrectly allocated, on the correct nodes using annotations available in the programming models. For the SOR code this can be done by specifying a blocked memory distribution corresponding to the matrix subdivision into blocks and their assignment to processors. In order to verify the efficiency of this optimization, the modified version of the SOR code has been executed on a real NUMA cluster, along with two further codes, a Gaussian elimination (GAUSS) and a particle simulation (WATER-NSQUARED from Splash-2 [24]), which have been optimized in a similar way.

Both, the optimized and the transparent version of each code, have been executed on a 4 node Xeon 450 MHz cluster interconnected using the D320 SCI adapter cards by Dolphin ICS and the software setup described in Section 2. The results of these experiments are shown in Table 1 in terms of speed-up in comparison to a sequential execution. They show a rather poor performance when executed on a fully transparent memory layout with a significant slow-down. This picture changes drastically after the modification of the memory layout. Now a significant speedup can be observed, especially the SOR code with a speedup of over 3.1 on 4 processors. Also all other codes benefit significantly proving both the importance and the feasibility of the presented approach.

5. Related work

In recent years, monitoring approaches are increasingly investigated and some hardware monitors [10,12, 14] have been developed for tracing of interconnection traffic. None of them, however, is applied to improve the data locality of running programs. An example is the trace instrument at Trinity College Dublin [14]. It has been built for monitoring the SCI packets transfered across the network, as our hardware monitor does, but is intended only for the analysis of the interconnect traffic with the goal to improve the modeling accuracy for network simulation systems.

Data locality, on the other hand, has been addressed intensively since it has a severe influence on performance of NUMA systems. Among these efforts [2, 4,11,15,16,19,22], which are primarily based on compiler analysis and page migration. One is especially closely related to the approach presented here. This is the *Dprof* profiling tool [2] developed by SGI. *Dprof* samples a program during its execution and records the program's memory access information as a histogram file. This histogram can be manually plotted using *gnuplot* for analyzing which regions of virtual memory each thread of a program accesses and further directs the explicit placement of specific ranges of virtual memory on a particular node.

In comparison with the approach presented in this paper, the *Dprof* report is based on statistical sampling and does therefore not record all references; in addition, numbers of memory accesses are shown at page granularity, allowing no detailed understanding of accesses within a single page. This restricts the accuracy of memory behavior analysis and prohibits a proper specification of an optimal data layout.

6. Conclusions

The successful deployment of NUMA architectures using the shared memory paradigm depends greatly on the efficient use of memory locality. Otherwise appli-

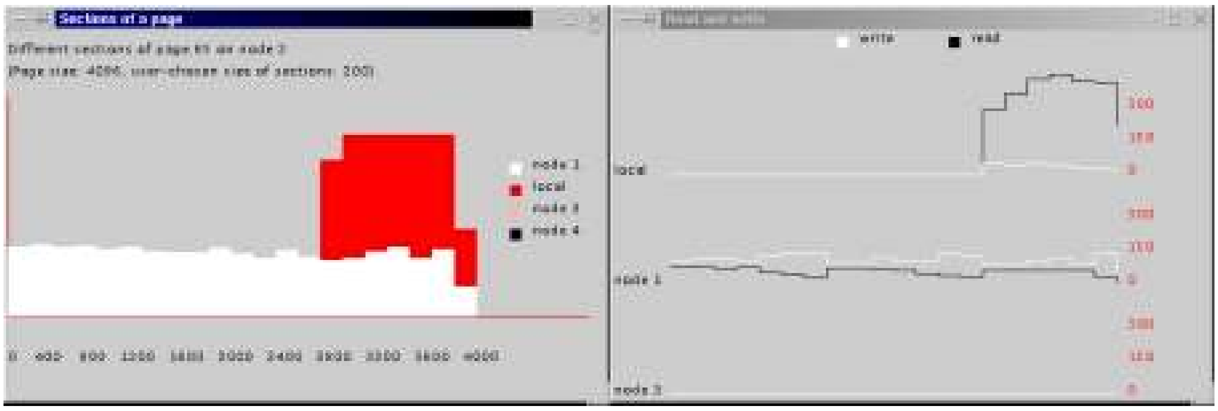


Fig. 4. The detailed access character of Page 65 on node 2 (SOR code).

Table 1
Execution on a real cluster with 4 nodes with and without optimizations

	SOR		GAUSS		WATER-N.	
	Time	Speedup	Time	Speedup	Time	Speedup
Sequential execution	1.36 s	1	4.83 s	1	15.83 s	1
Transparent parallel	78.16 s	0.0174	44.95 s	0.1075	116.12 s	0.14
Optimized parallel	0.43 s	3.16	2.11 s	2.289	5.20 s	3.04

cations will be penalized by excessive remote memory accesses and their significantly higher latencies. The tuning of applications towards this goal, however, is a difficult and complex task, since all communication is executed implicitly by read and write accesses and cannot directly be observed using simple software instrumentation without incurring a high probe overhead. Therefore, a low-level hardware monitoring facility in coordination with a comprehensive toolset has to be provided enabling users to perform the required optimizations.

Such an environment has been presented in this work. It consists of a low-level hardware monitor capable of observing the complete inter-node memory access traffic across the interconnection network and a tool infrastructure transforming the gathered information about the runtime behavior of the application into a human-readable way and enhancing it by additional information acquired through the various layers of the runtime environment. The current focus of this tool environment is a comprehensive visualization tool, the *Data Layout Visualizer* (*DLV*), which enables the presentation of the acquired information in a graphical and easy-to-use way. This includes the creation of memory access histograms which give a complete overview of the application execution across the whole address space without requiring any previous knowledge about the application. The information can then be used to analyze the memory access behavior of the target application and to optimize its memory distribution leading to a, in most cases significant, performance improvement. This has been proven by a set of numerical kernels, for which the optimization has enabled good speedup values on a 4-node SCI clusters. It is expected that a similar benefit can also be achieved for larger codes and systems as well as with more complex access patterns.

Even though this work was based on a single specific NUMA architecture, PC-based clusters interconnected with SCI, the approach is principally applicable to any other NUMA system, which allows to snoop memory traffic on any node. It therefore presents a general approach for the optimization of shared memory applications in such systems and can potentially be used far beyond the context of the SMiLE project.

References

[1] M. Bubak, W. Funika, R. Gembarowski and R. Wismüller, OMIS-compliant monitoring system for MPI applications, *Proc. 3rd International Conference on Parallel Processing and Applied Mathematics – PPAM'99*, Kazimierz Dolny, Poland, Sept. 1999, pp. 378–386,

[2] D. Cortesi, *Origin2000 and Onyx2 Performance Tuning and Optimization Guide*, chapter 4, Silicon Graphics Inc., 1998, Available from: http://techpubs.sgi.com:80/library/manuals/3000/007-3430-002/pdf/007-3430-002.pdf.

[3] H. Hellwagner and A. Reinefeld, *SCI: Scalable Coherent Interface: Architecture and Software for High-Performance Computer Clusters*, Volume 1734 of Lecture Notes in Computer Science, Springer Verlag, 1999.

[4] G. Howard and D. Lowenthal, An Integrated Compiler/Run-Time System for Global Data Distribution in Distributed Shared Memory Systems, *Proceedings of the Second Workshop on software Distributed Shared Memory Systems*, 2000.

[5] IEEE Computer Society, *IEEE Std 1596–1992: IEEE Standard for Scalable Coherent Interface*, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA, August, 1993.

[6] W. Karl, M. Leberecht and M. Schulz, Optimizing Data Locality for SCI–based PC–Clusters with the SMiLE Monitoring Approach, *Proceedings of International Conference on Parallel Architectures and Compilation Techniques* (*PACT*), Oct. 1999, pp. 169–176.

[7] W. Karl, M. Leberecht and M. Schulz, Supporting Shared Memory and Message Passing on Clusters of PCs with a SMiLE, in: *Proceedings of Workshop on Communication and Architectural Support for Network based Parallel Computing* (*CANPC*) (*held in conjunction with HPCA*), volume 1602 of LNCS, A. Sivasubramaniam and M. Lauria, eds, Springer Verlag, Berlin, 1999, pp. 196–210.

[8] W. Karl and M. Schulz, Hybrid-DSM: An Efficient Alternative to Pure Software DSM Systems on NUMA Architectures, *Proceedings of the 2nd International Workshop on Software DSM* (*held together with ICS 2000*), May 2000.

[9] W. Karl, M. Schulz and J. Trinitis, Multilayer Online-Monitoring for Hybrid DSM systems on top of PC clusters with a SMiLE, *Proceedings of 11th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 1786, of LNCS, Springer Verlag, Berlin, Mar. 2000, pp. 294–308.

[10] S. Karlin, D. Clark and M. Martonosi, SurfBoard-A Hardware Performance Monitor for SHRIMP, Technical Report TR-596-99, Princeton University, Mar. 1999.

[11] A. Krishnamurthy and K. Yelick, Analyses and Optimization for Shared Space Programs, *Journal of Parallel and Distributed Computation* **38**(2) (1996), 130–144.

[12] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta and J. Hennessy, The DASH Prototype: Logic Overhead and Performance, *IEEE Transactions on Parallel and Distributed Systems* **4**(1) (Jan. 1993), 41–61.

[13] M. Schulz, Efficient deployment of shared memory models on clusters of PCs using the SMiLEing HAMSTER approach, in: *Proceedings of the 4th International Conference on Algorithms and Architectures for Parallel Processing* (*ICA3PP*), A. Goscinski, H. Ip, W. Jia and W. Zhou, eds, World Scientific Publishing, Dec. 2000, pp. 2–14.

[14] M. Manzke and B. Coghlan, Non-intrusive deep tracing of SCI interconnect traffic, *Conference Proceedings of SCI Europe'99, a conference stream of Euro-Par'99*, Toulouse, France, Sept. 1999, pp. 53–58.

[15] A. Navarro and E. Zapata, An Automatic Iteration/Data Distribution Method based on Access Descriptors for DSMM, *Proceedings of the 12th International workshop on Languages and Compilers for Parallel Computing* (*LCPC'99*), San Diego, La Jolla, CA, USA, 1999.

[16] D. Nikolopoulos, T. Papatheodorou, et al., User-Level Dynamic Page Migration for Multiprogrammed Shared-Memory Multiprocessors, *Proceedings of the 29th International Conference on Parallel Processing*, Toronto, Canada, Aug. 2000, pp. 95–103.

[17] M. Schulz, *True shared memory programming on SCI-based clusters*, chapter 17, volume 1734 of Hellwagner and Reinefeld [3], Oct. 1999, pp. 291–311.

[18] M. Schulz, Efficient Coherency and Synchronization Management in SCI based DSM systems, in: *Proceedings of SCI-Europe 2000, The 3rd international conference on SCI–based technology and research*, G. Horn and W. Karl, eds, ISBN: 82-595-9964-3, Also available at http://wwwbode.in.tum.de/events/, Aug. 2000, pp. 31–36.

[19] S. Tandri and T. Abdelrahman, Automatic Partitioning of Data and Computations on Scalable Shared Memory Multiprocessors, *Proceedings of the 1997 International Conference on Parallel Processing* (*ICPP '97*), Washington – Brussels – Tokyo, Aug. 1997, pp. 64–73.

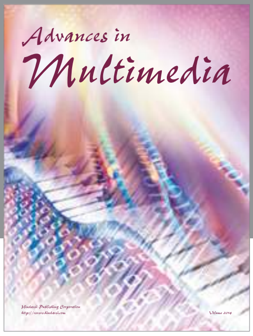
[20] J. Tao, W. Karl and M. Schulz, Using Simulation to Understand the Data Layout of Programs, *Proceedings of the IASTED International Conference on Applied Simulation and Modelling* (*ASM 2001*), page to appear, Marbella, Spain, Sept. 2001.

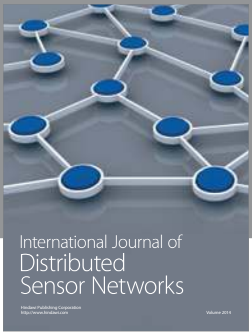
[21] J. Tao and W. Karl and M. Schulz, Visualizing the memory access behavior of shared memory applications on NUMA architectures, *Proceedings of the 2001 International Conference on Computational Science* (*ICCS*), volume 2074 of LNCS, San Francisco, CA, USA, May 2001, pp. 861–870.

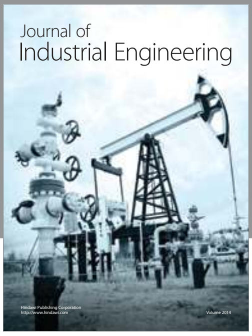
[22] B. Verghese, S. Devine, A. Gupta, M. Rosenblum, OS support for improving data locality on CC-NUMA compute servers, Technical Report CSL-TR-96-688, Computer System Laboratory, Stanford University, Feb. 1996.

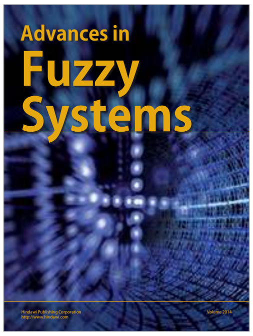
[23] R. Wismüller, Interoperability Support in the Distributed Monitoring System OCM, *Proc. 3rd International Conference on Parallel Processing and Applied Mathematics – PPAM'99*, Kazimierz Dolny, Poland, Sept. 1999, pp. 77–91.

[24] S. Woo, M. Ohara, E. Torrie, J. Singh and A. Gupta, The SPLASH–2 Programs: Characterization and Methodological Considerations, *Proceedings of the 22nd International Symposium on Computer Architecture* (*ISCA*), June 1995, pp. 24–36.









Hindawi

Submit your manuscripts at
http://www.hindawi.com